

**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**  
**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**  
**KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG**



**DOCKER HIGH AVAILABILITY AND DEMO**

**Giảng viên hướng dẫn:** ThS.Nguyễn Khánh Thuật

**Sinh viên 1:** Bùi Duy Long – 19520692

**Sinh viên 2:** Lương Nguyễn Mai Ly – 19520707

**Sinh viên 3:** Trần Thái Tuấn Anh - 19521219

**Sinh viên 4:** Nguyễn Mỹ Báo - 19521250

**TP. HỒ CHÍ MINH, 2021**  
**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN  
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG**



**DOCKER HIGH AVAILABILITY AND DEMO**

**Giảng viên hướng dẫn:** ThS.Nguyễn Khánh Thuật

**Sinh viên 1:** Bùi Duy Long – 19520692

**Sinh viên 2:** Lương Nguyễn Mai Ly – 19520707

**Sinh viên 3:** Trần Thái Tuấn Anh – 19521219

**Sinh viên 4:** Nguyễn Mỹ Báo - 19521250

**TP. HỒ CHÍ MINH, 2021**

## LỜI MỞ ĐẦU

Một mô hình máy chủ thông thường bao gồm máy chủ vật lý, hệ điều hành và ứng dụng. Do đó có nhược điểm là lãng phí tài nguyên vì một máy chủ chỉ cài đặt được một hệ điều hành, không tận dụng hết lợi thế bộ nhớ lẫn Ram.

Khắc phục hiện trạng trên thì công nghệ ảo hóa virtualization ra đời, điều này giúp một máy chủ vật lý có thể cài đặt được nhiều hệ điều hành, tận dụng tài nguyên tốt hơn. Tuy nhiên, việc khởi động máy ảo khá lâu, cần cấu hình cung cấp ngay từ tài nguyên ổ cứng và Ram thật cho máy ảo, lãng phí tài nguyên cho máy ảo. Từ đó, công nghệ containerlization ra đời. Với công nghệ này, trên một máy chủ vật lý, sẽ cài đặt được nhiều máy ảo (giống với công nghệ ảo hóa virtualization), nhưng tốt hơn ở chỗ là các máy con này (Guest OS) đều dùng chung phần nhân của máy mẹ (Host OS) và chia sẻ với nhau tài nguyên máy mẹ.

Việc setup và deploy application lên một hoặc nhiều server rất vất vả từ việc phải cài đặt các công cụ, môi trường cần cho application đến việc chạy được ứng dụng chưa kể việc không đồng nhất giữa các môi trường trên nhiều server khác nhau. Chính vì lý do đó Docker được ra đời để giải quyết vấn đề này.

# MỤC LỤC

LỜI MỞ ĐẦU .....	3
MỤC LỤC .....	4
DANH MỤC HÌNH ẢNH, BẢNG BIỂU .....	7
DANH MỤC KÝ HIỆU, CÔNG THỨC, THUẬT NGỮ .....	8
CHƯƠNG I - TỔNG QUAN VỀ DOCKER/HA .....	9
I.1 Nền tảng Docker .....	9
I.2 High Availability .....	9
I.3 Ứng dụng Docker.....	10
I.3.1 Phân phối nhanh chóng, nhất quán các ứng dụng.....	10
I.3.2 Triển khai đáp ứng và mở rộng quy mô.....	11
I.3.3 Chạy nhiều khối lượng công việc hơn trên cùng một phần cứng.....	11
I.4 Kiến trúc Docker.....	11
I.4.1 Docker Daemon .....	12
I.4.2 Docker Client .....	12
I.4.3 Docker Registry .....	12
I.4.4 Docker Object .....	13
I.5 Công nghệ cơ bản của Docker .....	14
CHƯƠNG II - DOCKER CƠ BẢN .....	15
II.1 Một số nội dung cơ bản trong docker.....	15
II.1.1 Docker dashboard .....	15
II.1.2 Docker Container.....	17
II.1.3 Docker Image .....	17
II.2 Docker với ứng dụng cơ bản .....	17
II.2.1 Running Application .....	18
II.2.2 Updating Application .....	20
II.2.3 Sharing Application.....	23
II.2.4 Persistent Database .....	24

TÀI LIỆU THAM KHẢO.....27



## DANH MỤC HÌNH ẢNH, BẢNG BIỂU

### Danh mục hình:

Hình I-1 Kiến trúc Docker .....	12
Hình II-1 Docker Dashboard.....	15
Hình II-2 Docker Tutorials.....	17
Hình II-3 Ứng dụng todo list trong VS Code.....	18
Hình II-4 Dockerfile.....	18
Hình II-5 Xây dựng ứng dụng từ Dockerfile .....	19
Hình II-6 Ứng dụng Todo App .....	20
Hình II-7 Container của todo app.....	20
Hình II-8 Docker ps.....	21
Hình II-9 Docker stop .....	21
Hình II-10 Docker rm.....	21
Hình II-11 Sử dụng Docker Dashboard để thao tác container .....	22
Hình II-12 Todo App update .....	22
Hình II-13 Create Repository .....	23
Hình II-14 Docker push lỗi .....	23
Hình II-15 Docker Push .....	24
Hình II-16 Sử dụng Docker Dashboard để thao tác container .....	25
Hình II-17 Terminal .....	25
Hình II-18 Terminal kết quả.....	26

### Danh mục bảng:

## DANH MỤC KÝ HIỆU, CÔNG THỨC, THUẬT NGỮ

**Thuật ngữ:**

<b>Viết Tắt</b>	<b>Ý nghĩa</b>	<b>Mô tả</b>
CI/CD	Continuous Integration and Continuous Delivery	Tích hợp liên tục và quy trình phân phối liên tục.



# CHƯƠNG I - TỔNG QUAN VỀ DOCKER/HA

Docker là một nền tảng mở để phát triển (develop), vận chuyển (shipping) và chạy (running) các ứng dụng. Docker cho phép tách các ứng dụng khỏi cơ sở hạ tầng để có thể phân phối phần mềm một cách nhanh chóng. Bằng cách tận dụng Docker để vận chuyển, thử nghiệm và triển khai mã một cách nhanh chóng từ đó có thể giảm đáng kể độ trễ giữa quá trình viết mã và chạy mã trong sản xuất.

## I.1 Nền tảng Docker

Docker cung cấp khả năng đóng gói và chạy một ứng dụng trong một môi trường cô lập được gọi là container. Sự cô lập và bảo mật cho phép chạy nhiều container đồng thời trên một máy chủ nhất định. Các container đó nhẹ và chứa mọi thứ cần thiết để chạy ứng dụng, vì vậy không cần phải dựa vào những gì hiện được cài đặt trên máy chủ lưu trữ. Có thể dễ dàng chia sẻ container trong khi làm việc và đảm bảo rằng mọi đối tượng chia sẻ đều có cùng container hoạt động theo cùng một cách.

Docker cung cấp công cụ và nền tảng để quản lý vòng đời của các container:

- Phát triển ứng dụng và các thành phần hỗ trợ ứng dụng bằng cách sử dụng các container.
- Container trở thành đơn vị phân phối và thử nghiệm ứng dụng.
- Triển khai ứng dụng vào môi trường sản xuất, dưới dạng một container hoặc một dịch vụ được điều phối. Điều này hoạt động giống nhau cho dù môi trường sản xuất là trung tâm dữ liệu cục bộ, nhà cung cấp đám mây hay kết hợp cả hai.

## I.2 High Availability

Nhắc đến High Availability thì đầu tiên ta cần nói đến Docker Swarm. Docker Swarm là một nhóm máy vật lý hoặc máy ảo đang chạy ứng dụng Docker và đã được định cấu hình để kết hợp với nhau thành một cụm. Khi một nhóm máy đã được nhóm lại với nhau, bạn vẫn có thể chạy các lệnh Docker mà bạn đã quen, nhưng giờ chúng sẽ được thực hiện bởi các máy trong cụm của bạn. Các hoạt động của cụm được kiểm soát bởi một trình quản lý bầy đàn và các máy đã tham gia vào cụm được gọi là các nút.

Docker Swarm được dùng như là một công cụ điều phối container, có nghĩa là nó cho phép người dùng quản lý được tất cả các containers đang được triển khai trên nhiều máy chủ. Nhờ vào việc nó có thể thiết lập và quản lý một cụm các nút Docker như

một hệ thống ảo duy nhất. Trong ngữ cảnh của Swarm, một cụm là một nhóm các máy chủ Docker hoạt động giống như một máy chủ Docker lớn duy nhất. Thay vì gặp rắc rối khi phải quyết định máy chủ lưu trữ nào sẽ bắt đầu mọi container, chúng ta có thể yêu cầu Swarm khởi động các container mà chúng ta cần sử dụng. Đây là tính năng đặc biệt quan trọng đối với các containers, vì nó tạo ra một nhóm hệ thống hợp tác từ đó có thể cung cấp dự phòng, cho phép chuyển đổi qua lại giữa các Docker Swarm nếu một hoặc nhiều nút gặp sự cố dẫn đến ngừng hoạt động. Một cụm Docker Swarm cũng cung cấp cho quản trị viên và nhà phát triển khả năng thêm hoặc bớt các lần lặp lại container khi nhu cầu tính toán có sự thay đổi.

Dựa vào nền tảng đó ta có tính khả dụng cao (HA) chính là việc luôn giữ cho các containers ở trạng thái sẵn sàng, ngay cả khi hỏng hóc. Điều này có thể thực hiện được bằng cách tạo nhiều phiên bản của cùng một vùng chứa. ... Để tạo vùng chứa có tính khả dụng cao trong Docker Swarm, chúng ta cần triển khai dịch vụ docker cho swarm với nginx image.

Thực hiện tạo nginx image bằng dòng lệnh

```
# docker service create --name nginx --publish 80:80 nginx
```

## **I.3 Ứng dụng Docker**

### ***I.3.1 Phân phối nhanh chóng, nhất quán các ứng dụng.***

Docker hợp lý hóa vòng đời phát triển bằng cách cho phép các nhà phát triển làm việc trong môi trường tiêu chuẩn hóa bằng cách sử dụng các container cung cấp các ứng dụng và dịch vụ. Các container là lựa chọn tuyệt vời để tích hợp liên tục và quy trình phân phối liên tục (Continuous Integration and Continuous Delivery - CI / CD).

Ví dụ:

- Các nhà phát triển viết mã và chia sẻ công việc của họ với đồng nghiệp bằng cách sử dụng Docker container.
- Họ sử dụng Docker để đẩy các ứng dụng của họ vào môi trường thử nghiệm và thực hiện các thử nghiệm tự động và thủ công.
- Khi các nhà phát triển tìm thấy lỗi, họ có thể sửa chúng trong môi trường phát triển và triển khai lại chúng vào môi trường thử nghiệm để kiểm tra và xác nhận.
- Khi quá trình thử nghiệm hoàn tất, việc nhận bản sửa lỗi cho khách hàng cũng đơn giản như đẩy image cập nhật vào môi trường sản xuất.

### ***1.3.2 Triển khai đáp ứng và mở rộng quy mô***

Nền tảng dựa trên container của Docker cho phép khối lượng công việc có tính di động cao. Bộ chứa Docker có thể chạy trên máy tính xách tay cục bộ của nhà phát triển, trên máy vật lý hoặc máy ảo trong trung tâm dữ liệu, trên các nhà cung cấp đám mây hoặc trong nhiều môi trường.

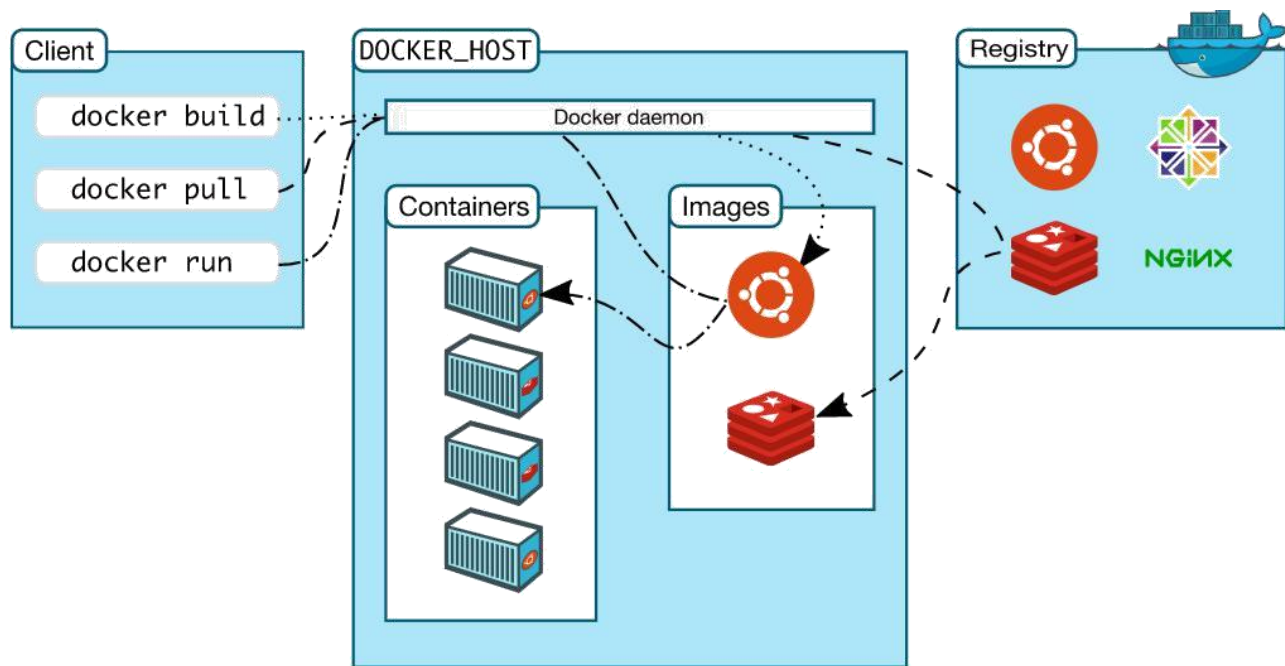
Tính di động và tính chất nhẹ của Docker cũng giúp dễ dàng quản lý động khối lượng công việc, mở rộng hoặc chia nhỏ các ứng dụng và dịch vụ khi nhu cầu kinh doanh yêu cầu, trong thời gian gần như thực.

### ***1.3.3 Chạy nhiều khối lượng công việc hơn trên cùng một phần cứng***

Docker nhẹ và nhanh. Nó cung cấp một giải pháp thay thế khả thi, hiệu quả về chi phí cho các máy ảo dựa trên hypervisor. Docker hoàn hảo cho các môi trường mật độ cao và cho các triển khai vừa và nhỏ.

## **1.4 Kiến trúc Docker**

Docker sử dụng kiến trúc máy khách-máy chủ (client- server). Ứng dụng Docker client giao tiếp với Docker daemon, docker daemon thực hiện công việc xây dựng, chạy và phân phối các Docker container. Docker client và daemon có thể chạy trên cùng một hệ thống hoặc có thể kết nối ứng dụng Docker client với Docker daemon từ xa. Ứng dụng Docker client và daemon giao tiếp bằng REST API, qua UNIX sockets hoặc network interface. Một ứng dụng Docker client khác là Docker Compose, cho phép làm việc với các ứng dụng bao gồm một tập hợp các container.



Hình 1-1 Kiến trúc Docker

(Nguồn: [Docker overview](#) | [Docker Documentation](#))

#### ***1.4.1 Docker Daemon***

Docker daemon (dockerd) lắng nghe các yêu cầu API Docker và quản lý các đối tượng Docker như containers, images, networks. Một daemon cũng có thể giao tiếp với các daemon khác để quản lý các dịch vụ Docker.

#### ***1.4.2 Docker Client***

Docker Client (docker) là cách chính mà nhiều người dùng Docker tương tác với Docker. Khi sử dụng các lệnh như docker run, client sẽ gửi các lệnh này đến dockerd để thực hiện chúng. Lệnh docker sử dụng API Docker. Docker client có thể giao tiếp với nhiều hơn một daemon.

#### ***1.4.3 Docker Registry***

Docker Registry lưu trữ các Docker Images. Docker Hub là cơ quan đăng ký công khai mà bất kỳ ai cũng có thể sử dụng và Docker được định cấu hình để tìm kiếm images trên Docker Hub theo mặc định.

Khi sử dụng lệnh docker pull hoặc docker run, các images cần thiết sẽ được lấy từ registry đã định cấu hình. Khi sử dụng lệnh docker push, images sẽ được đẩy vào registry.

#### ***I.4.4 Docker Object***

Khi sử dụng Docker, tức là đang tạo và sử dụng image, containers, networks, volumes, plugin và các đối tượng khác.

##### **I.4.4.1 - Images**

Một Images là mẫu chỉ đọc với các hướng dẫn để tạo Docker Container. Thông thường, một images dựa trên một images khác, với một số tùy chỉnh bổ sung.

Có thể tạo images của riêng hoặc bạn chỉ có thể sử dụng những hình ảnh do người khác tạo và xuất bản trong registry.

Để xây dựng images của riêng, tạo một Dockerfile với cú pháp đơn giản để xác định các bước cần thiết để tạo images và chạy nó. Mỗi lệnh trong Dockerfile sẽ tạo ra một layer trong images. Khi thay đổi Dockerfile và xây dựng lại images, chỉ những layer đã thay đổi mới được xây dựng lại. Đây là một phần lý do làm cho images trở nên nhẹ, nhỏ và nhanh, khi so sánh với các công nghệ ảo hóa khác.

##### **I.4.4.2 - Containers**

Containers là một thể hiện có thể chạy được của một image. Có thể tạo, bắt đầu, dừng, di chuyển hoặc xóa Container bằng API Docker hoặc CLI. Có thể kết nối container với một hoặc nhiều networks, đính kèm bộ nhớ vào nó hoặc thậm chí tạo images mới dựa trên trạng thái hiện tại của nó.

Theo mặc định, một container được cách ly tương đối tốt với các container khác và máy chủ của nó. Có thể kiểm soát mức độ biệt lập của network, bộ nhớ hoặc các hệ thống con cơ bản khác của container với các container khác hoặc với máy chủ.

Container được xác định bởi image của nó cũng như bất kỳ tùy chọn cấu hình nào cung cấp cho nó khi tạo hoặc khởi động nó. Khi một container bị xóa, mọi thay đổi đối với trạng thái của nó mà không được lưu trữ trong bộ nhớ liên tục sẽ biến mất.

##### **I.4.4.3 - Ví dụ về lệnh docker run**

Lệnh sau chạy một container Ubuntu, tương tác Command-Line và chạy /bin/bash:

```
$ docker run -i -t ubuntu /bin/bash
```

Khi chạy lệnh này:

- 1) Nếu không có images ubuntu, Docker sẽ kéo nó từ registry, như thể đã chạy “docker pull ubuntu” theo cách thủ công.
- 2) Docker tạo một container mới, như thể đã chạy lệnh “docker container create” theo cách thủ công.
- 3) Docker phân bổ hệ thống tệp đọc-ghi cho container, như layer cuối cùng của nó. Điều này cho phép một container đang chạy tạo hoặc sửa đổi các tệp và thư mục trong hệ thống tệp cục bộ.
- 4) Docker tạo giao diện mạng để kết nối container với mạng mặc định, vì không chỉ định bất kỳ tùy chọn mạng nào. Điều này bao gồm việc gán địa chỉ IP cho container. Theo mặc định, container có thể kết nối với các mạng bên ngoài bằng kết nối mạng của máy chủ.
- 5) Docker khởi động container và thực thi / bin / bash. Vì container đang chạy tương tác và được gắn vào thiết bị đầu cuối (do -i và -t), có thể cung cấp đầu vào bằng bàn phím trong khi đầu ra được ghi vào thiết bị đầu cuối.
- 6) Khi gõ exit để kết thúc lệnh / bin / bash, container sẽ dừng lại nhưng không bị xóa. Có thể bắt đầu lại hoặc xóa nó.

## **I.5 Công nghệ cơ bản của Docker**

Docker được viết bằng ngôn ngữ lập trình Go và tận dụng một số tính năng của nhân Linux để cung cấp chức năng của nó. Docker sử dụng một công nghệ được gọi là namespaces để cung cấp không gian làm việc biệt lập được gọi là container. Khi chạy một container, Docker tạo một tập hợp các không gian tên cho container đó.

Các namespaces này cung cấp một lớp cách ly. Mỗi khía cạnh của một container chạy trong một namespaces riêng biệt và quyền truy cập của nó bị giới hạn trong namespaces đó.

## CHƯƠNG II - DOCKER CƠ BẢN

### II.1 Một số nội dung cơ bản trong docker

#### II.1.1 Docker dashboard

Docker dashboard cung cấp một giao diện đơn giản cho phép bạn quản lý vùng chứa, ứng dụng và hình ảnh của mình trực tiếp từ máy tính của bạn mà không cần phải sử dụng CLI để thực hiện các tác vụ cốt lõi. Chế độ xem vùng chứa / ứng dụng cung cấp chế độ xem thời gian chạy của tất cả các vùng chứa và ứng dụng của bạn.

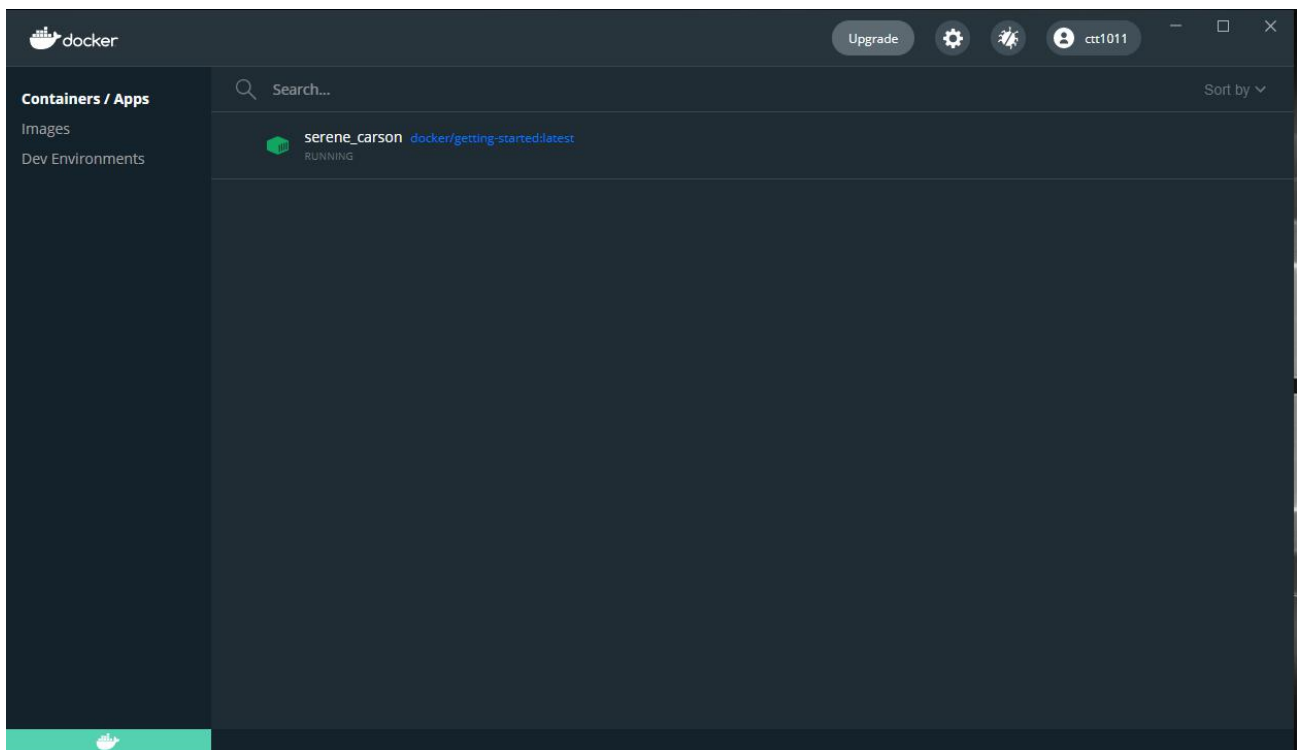
Chạy câu lệnh: `docker run -d -p 80:80 docker/getting-started`

- `-d` - chạy container ở chế độ tách rời (trong nền).
- `-p 80:80` - ánh xạ cổng 80 của máy chủ đến cổng 80 trong container.
- `docker / get-started` - images để sử dụng.

Có thể kết hợp các cờ ký tự đơn để rút ngắn lệnh đầy đủ. Ví dụ, lệnh trên có thể được viết như sau:

```
docker run -dp 80:80 docker/getting-started
```

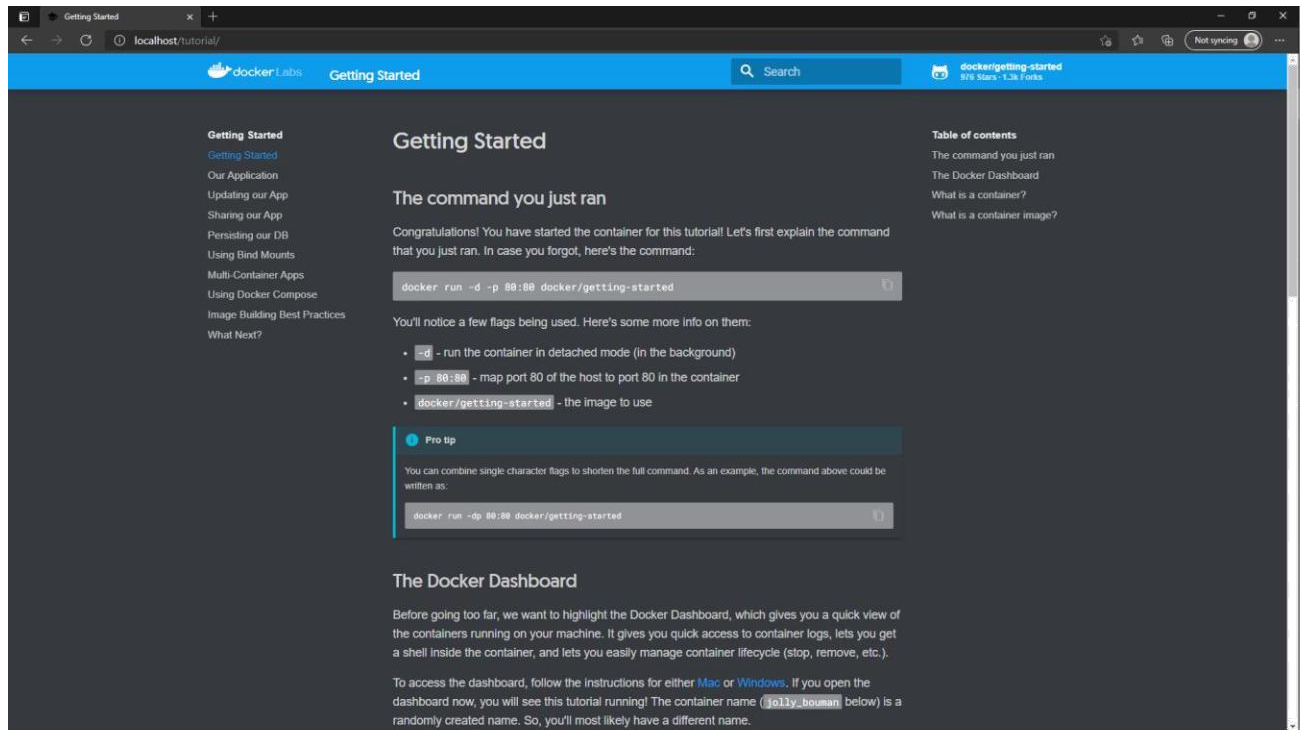
Docker Dashboard có sẵn cho Mac và Windows. Nếu mở dashboard ngay khi chạy lệnh trên sẽ thấy container đang chạy. Tên vùng chứa (`serene_carson` bên dưới) là tên được tạo ngẫu nhiên.



### *Hình II-1 Docker Dashboard*

Khi nhập địa chỉ localhost trên trình duyệt, sẽ nhận được một trang hướng dẫn về docker.





Hình II-2 Docker Tutorials

### II.1.2 Docker Container

Container được hiểu đơn giản là một môi trường để thực thi ứng dụng mà ở đó người dùng có thể chạy ứng dụng một cách độc lập với hệ thống. Những container này thường rất gọn nhẹ và cho phép bạn chạy ứng dụng trong đó rất nhanh chóng và dễ dàng. Hay có thể nói một cách khác thì Container là một quy trình khác trên máy đã được cách ly với tất cả các quy trình khác trên máy chủ. Sự cô lập đó thúc đẩy namespace và nhóm hạt nhân, những tính năng đã có trong Linux từ lâu. Docker đã làm việc để làm cho những khả năng này trở nên dễ tiếp cận và dễ sử dụng hơn đối với người dùng.

### II.1.3 Docker Image

Khi chạy một container, nó sử dụng một hệ thống tệp riêng biệt. Hệ thống tệp tùy chỉnh này được cung cấp bởi một container image. Vì image chứa hệ thống tệp của container nên nó phải chứa mọi thứ cần thiết để chạy một ứng dụng - tất cả các phụ thuộc, cấu hình, tập lệnh, mã nhị phân, v.v. Images cũng chứa cấu hình khác cho container, chẳng hạn như biến môi trường, lệnh mặc định để chạy, và siêu dữ liệu khác.

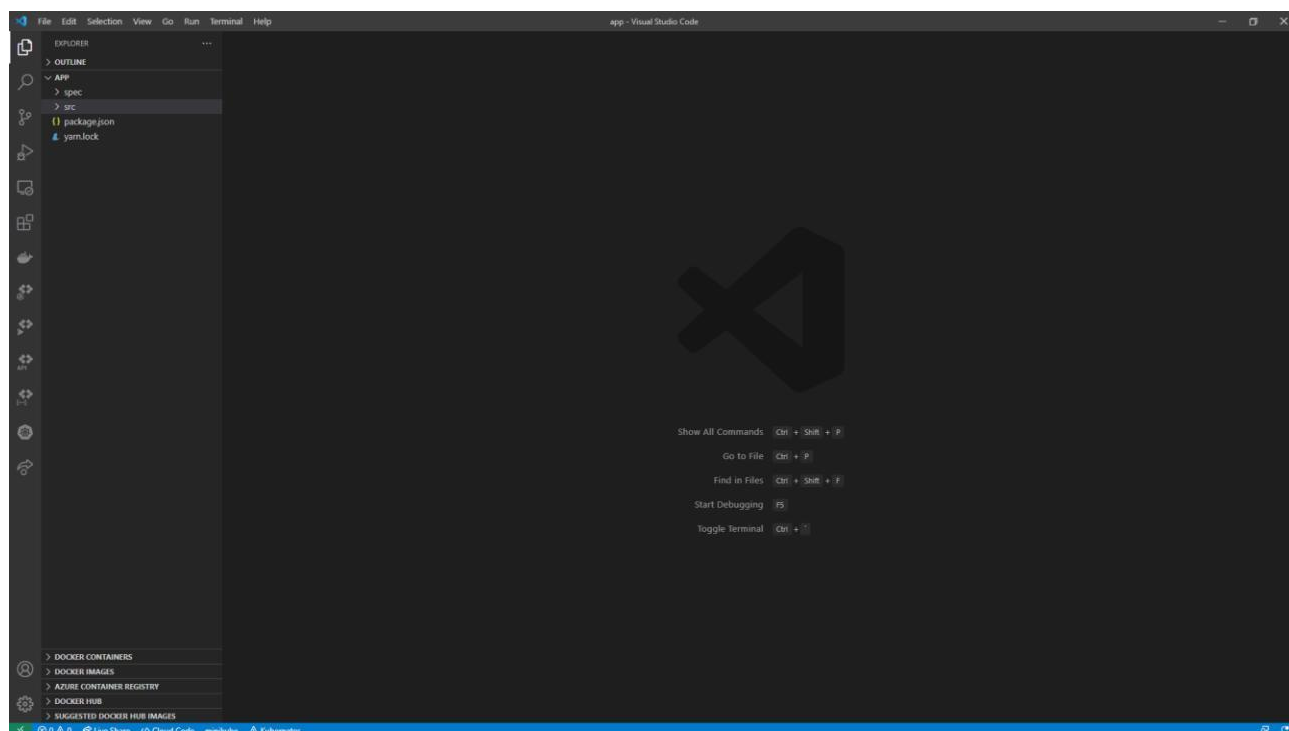
## II.2 Docker với ứng dụng cơ bản

## II.2.1 Running Application

### II.2.1.1 - Tải ứng dụng

Trước khi có thể chạy ứng dụng, cần lấy mã nguồn ứng dụng vào máy của mình. Chạy câu lệnh `docker run -dp 80:80 docker/getting-started` sau đó vào trình duyệt nhập địa chỉ `http://localhost/assets/app.zip`. Đây là ứng dụng quản lý những việc cần làm của Docker cung cấp.

Sau đó giải nén và mở bằng code editor (sử dụng Visual Studio Code).

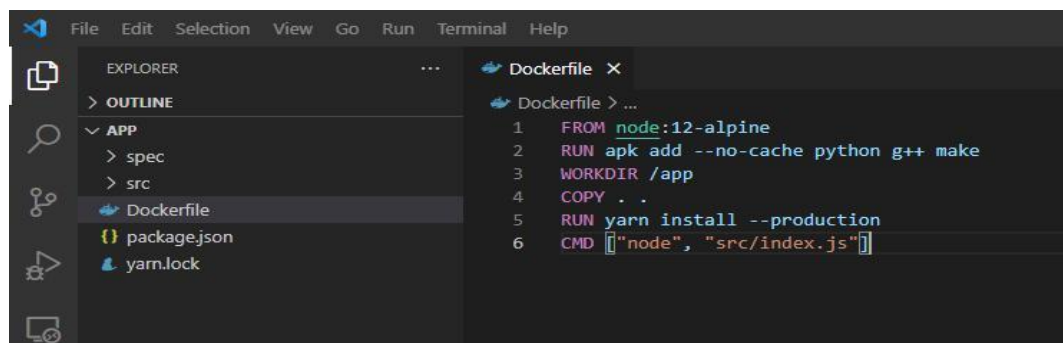


Hình II-3 Ứng dụng todo list trong VS Code

### II.2.1.2 - Xây dựng Image của ứng dụng

Để xây dựng ứng dụng, cần sử dụng Dockerfile. Dockerfile chỉ đơn giản là một kịch bản hướng dẫn dựa trên văn bản được sử dụng để tạo một container image.

Dockerfile không có extension như .txt.



## Hình II-4 Dockerfile

Mở Terminal và dùng lệnh:

`docker build -t getting-started .`

Câu lệnh này sẽ xây dựng một container image mới. Có nhiều layer được tải xuống do sử dụng FROM node:12-alpine image. Sau khi image được tải xuống, sao chép trong ứng dụng của mình và sử dụng yarn để cài đặt các phần phụ thuộc của ứng dụng. Chỉ thị CMD chỉ định lệnh mặc định để chạy khi bắt đầu một container từ image này.

Cuối cùng, cờ -t gắn thẻ image. Đây là tên có thể đọc được của con người cho image. Vì đặt tên cho image là getting-started, có thể tham chiếu đến image đó khi chạy một container.

Dấu “.” ở cuối lệnh docker build nói rằng Docker nên tìm kiếm Dockerfile trong thư mục hiện tại.

```
PS C:\Users\chung\Desktop\app> docker build -t getting-started .
[+] Building 43.7s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 183B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/node:12-alpine
=> [auth] library/node:pull token for registry-1.docker.io
=> [1/5] FROM docker.io/library/node:12-alpine@sha256:9923c9efb13cf7535f67e49b03010f0977a800060e4c8e0e2c93433a6bfa1e77
=> => resolve docker.io/library/node:12-alpine@sha256:9923c9efb13cf7535f67e49b03010f0977a800060e4c8e0e2c93433a6bfa1e77
=> sha256:9923c9efb13cf7535f67e49b03010f0977a800060e4c8e0e2c93433a6bfa1e77 1.43kB / 1.43kB
=> sha256:d714e8b527d784cd12b3dfc822f771c7f3531acb57f483be2c8f0997924a37df 1.16kB / 1.16kB
=> sha256:deae3752431af726a256aee99bbd35d9323de7ffa76ec42d4629b2751b8d11 6.73kB / 6.73kB
=> sha256:ddad3d7c1e9eadf9153f8921a7c9790f880a390163df453be1566e9ef0d546ee 2.82kB / 2.82kB
=> sha256:3a8370f05d5d4a08dbae2a139bd9286c42712a66ea263d39c0914d4a37eafe72 24.60kB / 24.60kB
=> sha256:71a8563b7feaa9ea167906666dd1173a08512f3f84acaa6dd4cd3261d15fbf9 2.24kB / 2.24kB
=> sha256:119c7e14957d3c5cab0ae2e0748bbd997a7a54f01b668d939e5bf656ad6dd585 281B / 281B
=> extracting sha256:ddad3d7c1e9eadf9153f8921a7c9790f880a390163df453be1566e9ef0d546ee
=> extracting sha256:3a8370f05d5d4a08dbae2a139bd9286c42712a66ea263d39c0914d4a37eafe72
=> extracting sha256:71a8563b7feaa9ea167906666dd1173a08512f3f84acaa6dd4cd3261d15fbf9
=> extracting sha256:119c7e14957d3c5cab0ae2e0748bbd997a7a54f01b668d939e5bf656ad6dd585
=> [internal] load build context
=> => transferring context: 4.63kB
=> [2/5] RUN apk add --no-cache python g++ make
=> [3/5] WORKDIR /app
=> [4/5] COPY . .
=> [5/5] RUN yarn install --production
=> exporting to image
=> exporting layers
=> writing image sha256:08369d7bfba08e32babb28e2b533c436b3f5cae65defdb68dd2983e6a9a24233
=> naming to docker.io/library/getting-started

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
PS C:\Users\chung\Desktop\app>
```

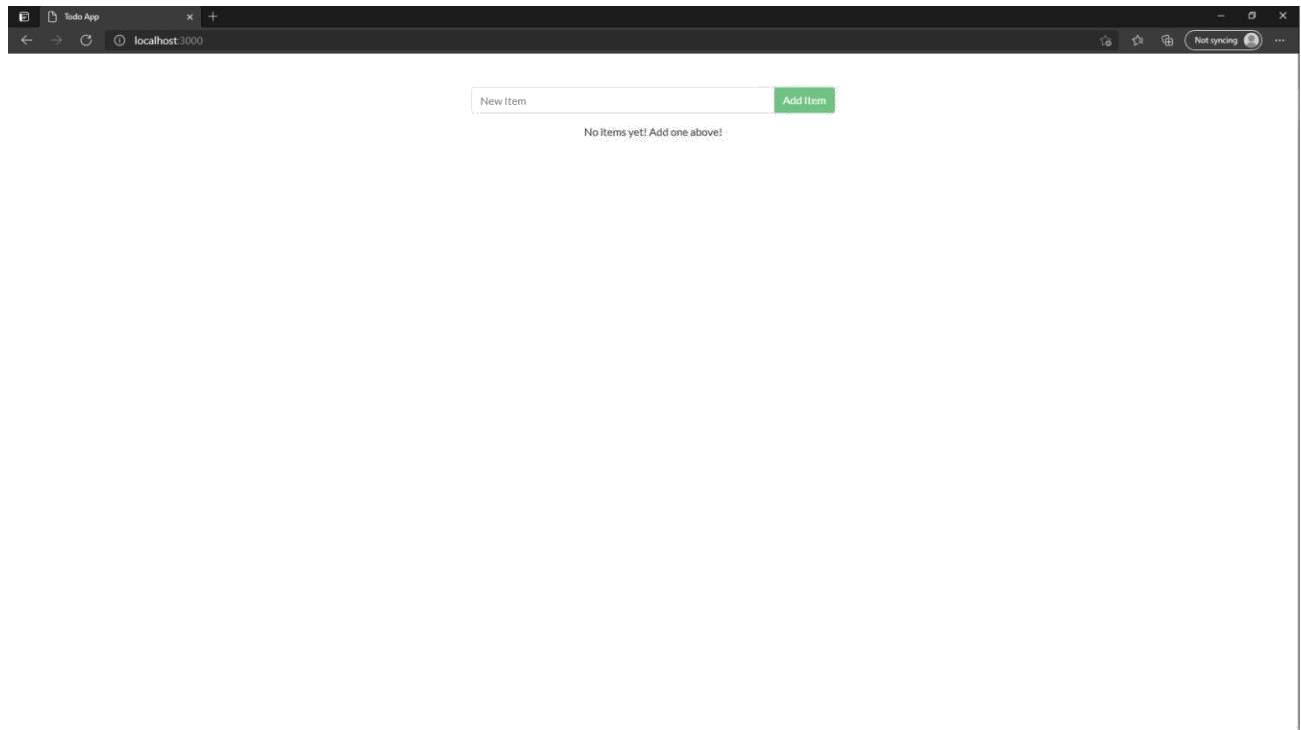
## Hình II-5 Xây dựng ứng dụng từ Dockerfile

### II.2.1.3 - Chạy ứng dụng

Sử dụng câu lệnh docker run để chạy ứng dụng:

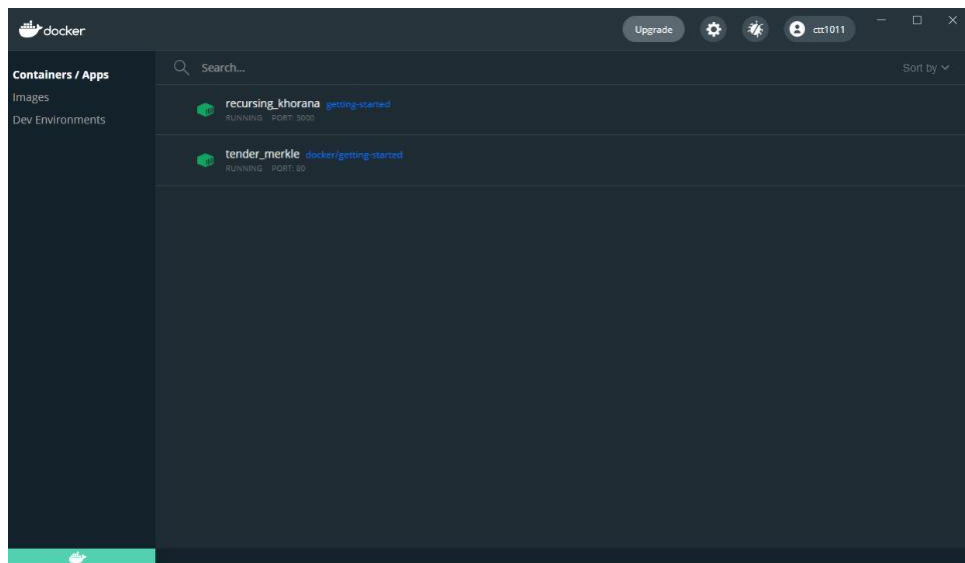
`docker run -dp 3000:3000 getting-started`

Mở trình duyệt và địa đến địa chỉ: localhost:3000 để chạy ứng dụng:



*Hình II-6 Ứng dụng Todo App*

Trong Docker Dashboard xuất hiện container đang chạy.



*Hình II-7 Container của todo app*

## ***II.2.2 Updating Application***

Cập nhật lại mã nguồn:

Tại `src/static/js/app.js` cập nhật dòng 56 nội dung sau:

<p className="text-center">You have no todo items yet! Add one above!</p>

Sau đó chạy câu lệnh:

```
docker build -t getting-started .
```

```
docker run -dp 3000:3000 getting-started
```

Sẽ nhận được lỗi như sau:

```
docker: Error response from daemon: driver failed programming external connectivity on endpoint strange_beaver (eb6c83da0a61663e005ba4e6caa51aabb3d7826568ffc344fb4a6cc2dcca7d7): Bind for 0.0.0.0:3000 failed: port is already allocated.
```

Vì không thể bắt đầu container mới vì container cũ vẫn đang chạy. Lý do đây là sự cố là vì container đó đang sử dụng cổng 3000 của máy chủ và chỉ có một tiến trình trên máy (bao gồm container) có thể lắng nghe một cổng cụ thể. Để khắc phục điều này, chúng ta cần loại bỏ container cũ.

Sử dụng docker ps để lấy ID container.

```
PS C:\Users\chung\Desktop\app> docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
3662c36f0bf2	08369d7bfba0	"docker-entrypoint.s..."	About an hour ago	Up About an hour	0.0.0.0:3000->3000/tcp	recursing_khorana
9211dbff6f91	docker/getting-started	"/docker-entrypoint..."	2 hours ago	Up 2 hours	0.0.0.0:80->80/tcp	tender_merkle

Hình II-8 Docker ps

Sử dụng docker stop <container – id> để dừng container.

```
PS C:\Users\chung\Desktop\app> docker stop 3662c36f0bf2
3662c36f0bf2
```

Hình II-9 Docker stop

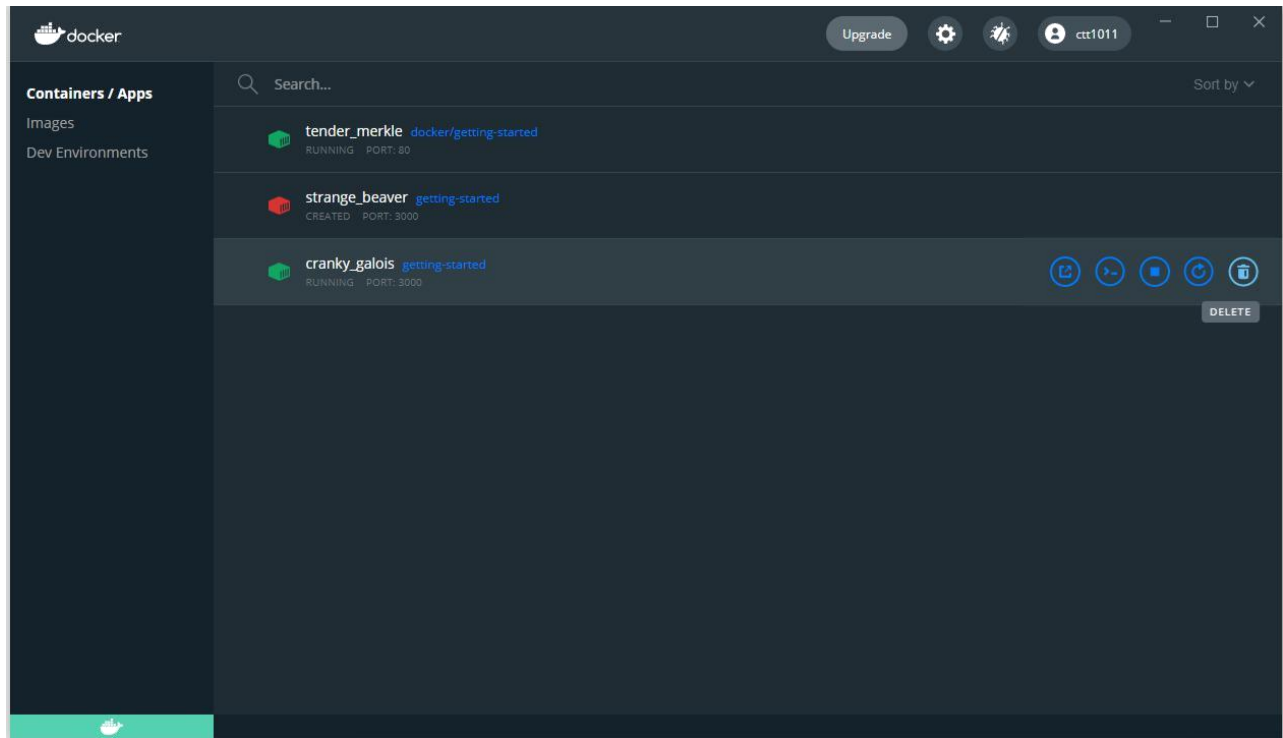
Sử dụng docker rm <container-id> để loại bỏ container.

```
PS C:\Users\chung\Desktop\app> docker rm 3662c36f0bf2
3662c36f0bf2
```

Hình II-10 Docker rm

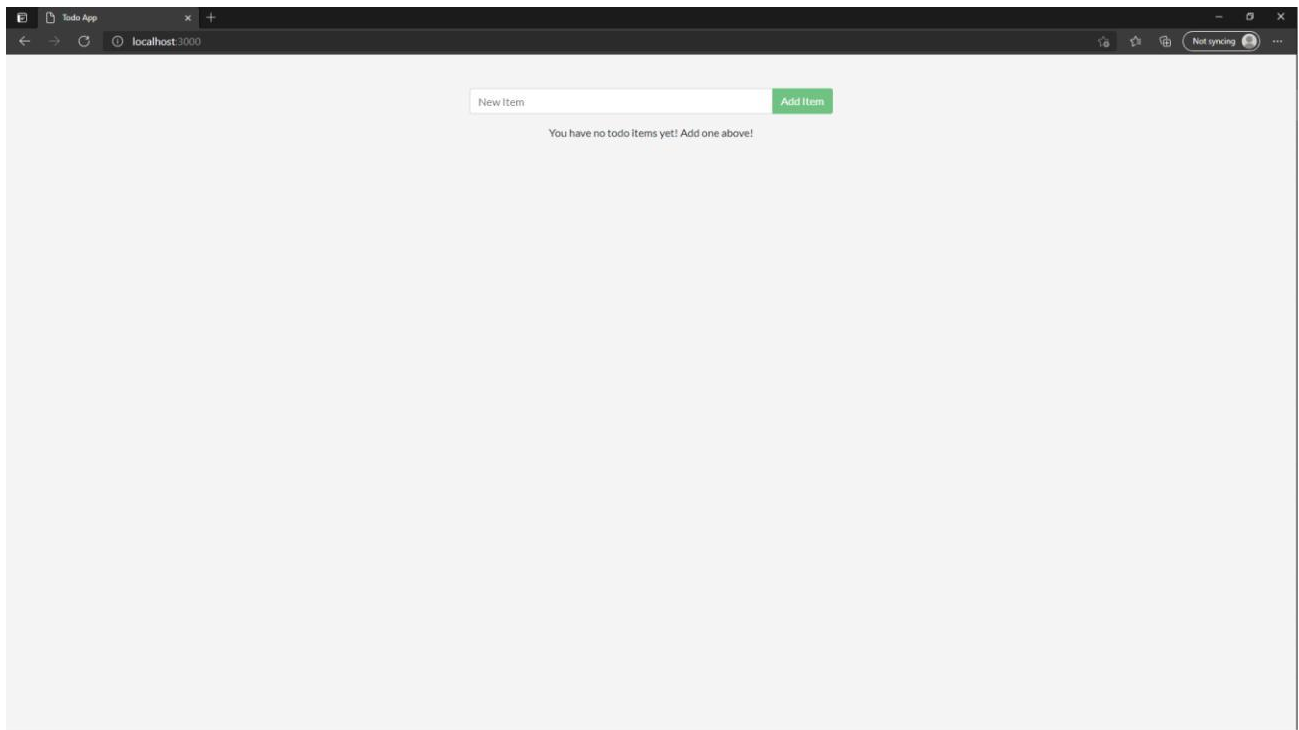
Có thể dừng và xóa container trong một lệnh bằng cách thêm cờ "force" vào lệnh docker rm. Ví dụ: docker rm -f <the-container-id>.

Hoặc có thể sử dụng Docker Dashboard để xóa container vì giao diện đơn giản của nó.



*Hình II-11 Sử dụng Docker Dashboard để thao tác container*

Bây giờ có thể chạy câu lệnh: `docker run -dp 3000:3000 getting-started` và refresh trình duyệt để khởi động lại ứng dụng.



*Hình II-12 Todo App update*

## II.2.3 Sharing Application

### II.2.3.1 - Tạo Repository

Để đẩy lên một image cần tạo một Repo trên Docker Hub.

- Vào trang chủ Docker Hub và đăng nhập
- Chọn Create Repository.
- Sử dụng getting-started cho repo name và public.
- Chọn Create.

Thực hiện xong sẽ được giao diện như hình.



Hình II-13 Create Repository

### II.2.3.2 - Push Image

Sử dụng câu lệnh:

```
docker push ctt1011/getting-started
```

```
C:\Users\chung>docker push ctt1011/getting-started
Using default tag: latest
The push refers to repository [docker.io/ctt1011/getting-started]
An image does not exist locally with the tag: ctt1011/getting-started
```

Hình II-14 Docker push lỗi

Lệnh push đang tìm kiếm một image có tên là ctt1011/ get-started, nhưng không tìm thấy một image nào. Nếu chạy docker image ls, cũng sẽ không thấy.

Để khắc phục điều này, cần "gắn thẻ" cho image hiện có mà đã xây dựng để đặt tên khác cho nó.

Đăng nhập vào docker Hub sử dụng:

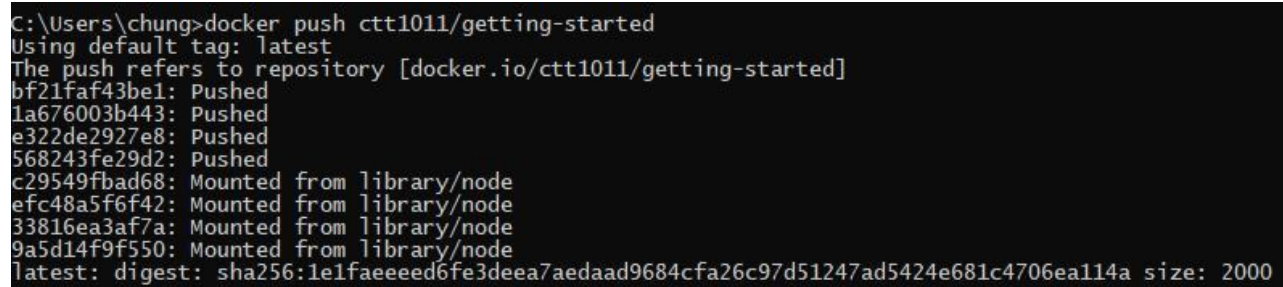
```
docker login -u YOUR-USER-NAME
```

Dùng docker tag cho getting-started tên mới.

```
docker tag getting-started YOUR-USER-NAME/getting-started
```

Dùng docker push lần nữa:

`docker push YOUR-USER_NAME/getting-started`



```
C:\Users\chung>docker push ctt1011/getting-started
Using default tag: latest
The push refers to repository [docker.io/ctt1011/getting-started]
bf21faf43be1: Pushed
1a676003b443: Pushed
e322de2927e8: Pushed
568243fe29d2: Pushed
c29549fbad68: Mounted from library/node
efc48a5f6f42: Mounted from library/node
33816ea3af7a: Mounted from library/node
9a5d14f9f550: Mounted from library/node
latest: digest: sha256:1e1faeeed6fe3deea7aedaad9684cfa26c97d51247ad5424e681c4706ea114a size: 2000
```

*Hình II-15 Docker Push*

## **II.2.4 Persistent Database**

### **II.2.4.1 - Hệ thống tập tin Container**

Khi một container chạy, nó sử dụng các layer khác nhau từ một image cho hệ thống tệp của nó. Mỗi container cũng có "khoảng trống" riêng để tạo / cập nhật / xóa tệp. Mọi thay đổi sẽ không được nhìn thấy trong container khác, ngay cả khi chúng đang sử dụng cùng một image.

Để xem điều này hoạt động, chúng ta sẽ bắt đầu hai container và tạo một tệp trong mỗi container. Những gì sẽ thấy là các tệp được tạo trong một container này không có sẵn trong một container khác.

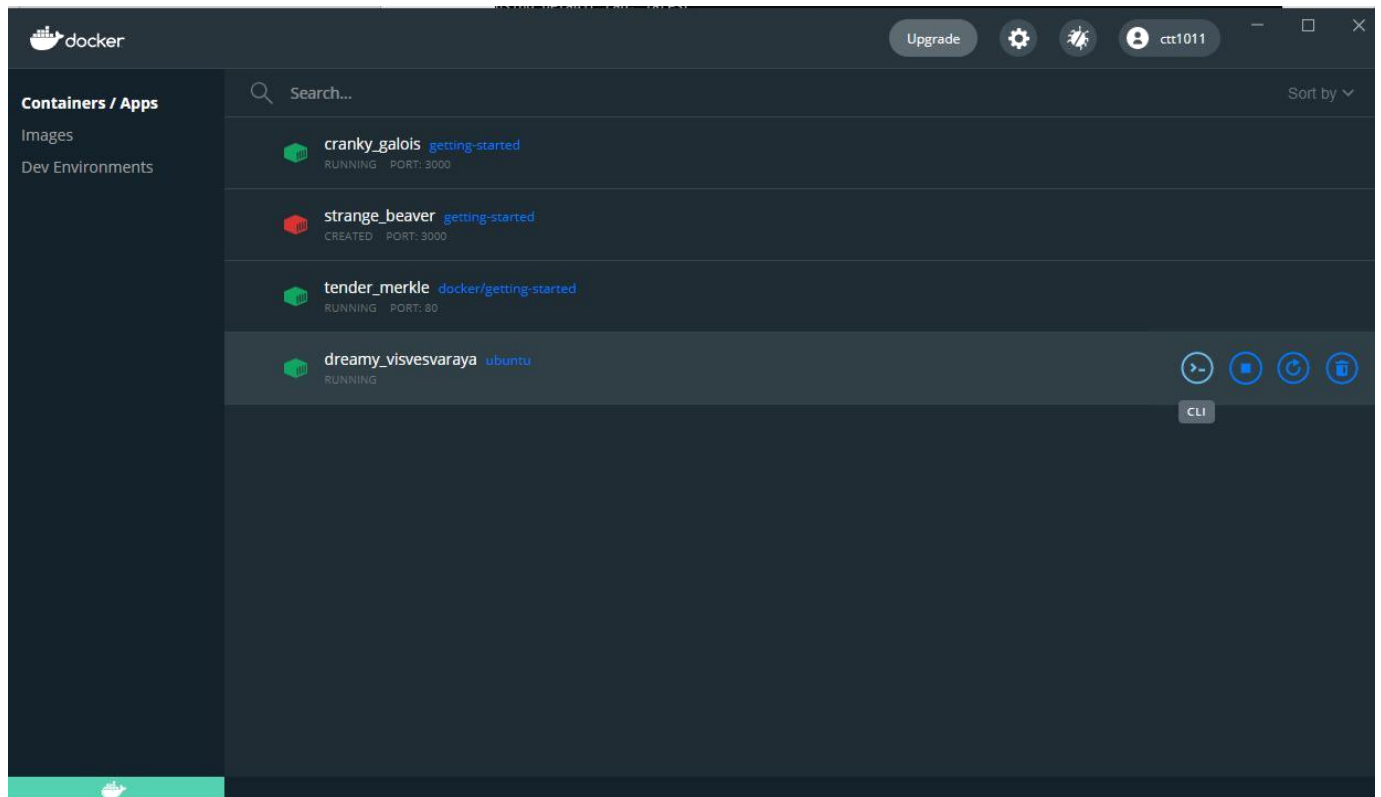
Khởi động container ubuntu sẽ tạo một tệp có tên /data.txt với một số ngẫu nhiên từ 1 đến 10000.

```
docker run -d ubuntu bash -c "shuf -i 1-10000 -n 1 -o /data.txt && tail -f /dev/null"
```

Lệnh trên sẽ bắt đầu một bash shell và gọi hai lệnh (&&). Phần đầu tiên chọn một số ngẫu nhiên và ghi nó vào /data.txt. Lệnh thứ hai chỉ đơn giản là xem một tệp để giữ cho container hoạt động.

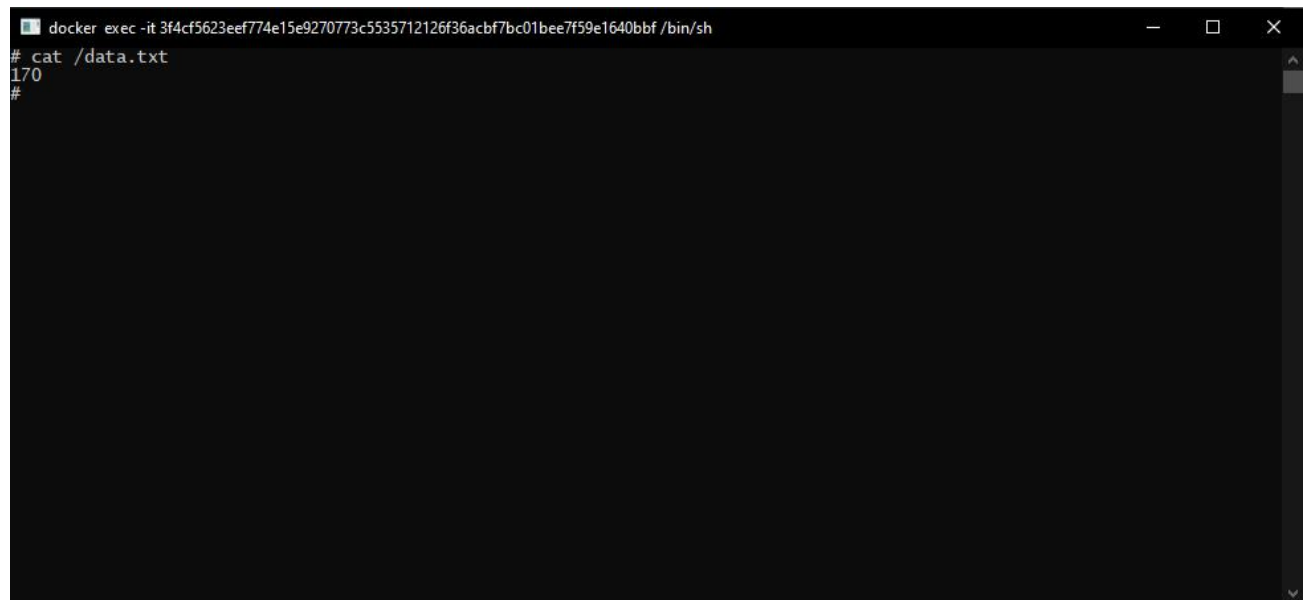
Có thể thấy đầu ra bằng cách thực thi vào vùng chứa. Để làm như vậy, hãy mở Docker Dashboard và nhấp vào hành động đầu tiên của container đang chạy image ubuntu.





Hình II-16 Sử dụng Docker Dashboard để thao tác container

Một Terminal xuất hiện, chạy lệnh sau để xem nội dung data.txt : `cat /data.txt`



Hình II-17 Terminal

Có thể sử dụng lệnh thực thi của docker để làm điều tương tự. Cần lấy ID của container (sử dụng `docker ps` để lấy) và lấy nội dung bằng lệnh sau.

```
docker exec <container-id> cat /data.txt
```

```
C:\Users\chung>docker exec 3f4cf5623eef cat /data.txt  
170
```

*Hình II-18 Terminal minh họa*

## **TÀI LIỆU THAM KHẢO**

1. Docker Documentation ([docs.docker.com/get-docker](https://docs.docker.com/get-docker)).